4. IMPLEMENTASI SISTEM

4.1. Perangkat Lunak yang digunakan

Aplikasi yang dibuat pada penelitian ini dibuat menggunakan Flutter SDK versi 3.22.2 dan bahasa pemrograman Dart versi 3.4.3 dengan Visual Studio Code sebagai *code editor*. Dan juga beberapa skrip yang menggunakan bahasa pemrograman Python dan Typescript.

4.2. Implementasi Program

Tabel 4.1

Hubungan Desain Sistem dan Segmen Program

Nama	Segmen Program	Activity Diagram
Manajemen Karyawan	Segmen Program 4.1 Segmen Program 4.2 Segmen Program 4.3	Alur Manajemen Karyawan
Manajemen User	Segmen Program 4.4 Segmen Program 4.5 Segmen Program 4.6	Alur Manajemen User
Absensi Admin	Segmen Program 4.7 Segmen Program 4.8 Segmen Program 4.9	Alur Absensi Karyawan Alur Absensi Cabutan
Absensi User	Segmen Program 4.10 Segmen Program 4.11 Segmen Program 4.12	
Perizinan	Segmen Program 4.13 Segmen Program 4.14 Segmen Program 4.15	Alur Perizinan
Pengumuman	Segmen Program 4.16 Segmen Program 4.17 Segmen Program 4.18	Alur Pembuatan Pengumuman

Gaji	Segmen Program 4.19 Segmen Program 4.20 Segmen Program 4.21 Segmen Program 4.22 Segmen Program 4.23	Alur Gaji
Help Chat	Segmen Program 4.24 Segmen Program 4.25 Segmen Program 4.26 Segmen Program 4.27	Alur *Help Chat*

4.2.1. Manajemen Karyawan

Segmen Program 4.1 Model karyawan

```
class EmployeeModel {
   final int id;
   String name;
   String? shortedName;
   String? initials;
   int? age;
   String? address;
   String? birthPlace;
   DateTime? birthDate;
   String? phoneNumber;
   String? gender;
   String? religion;
   int? attendanceMachineID;
   DateTime? entryDate;
   double? salary;
   PositionModel position;
   SalaryModel? thisMonthSalary;

   EmployeeModel({
          required this.id,
          required this.name,
          required this.shortedName,
          required this.initials,
          this.age,
          this.address = '',
          this.birthPlace,
          this.birthDate,
          this.phoneNumber = '',
          this.gender = '',
          this.religion = '',
          this.attendanceMachineID = 0,
          this.entryDate,
          this.salary = 0.0,
          required this.position,
          this.thisMonthSalary,
   });

   factory EmployeeModel.fromSupabase(Map<String, dynamic> employee) {
          SalaryModel? thisMonthSalary;
          String name = employee['nama'];
          List<String> nameParts = name.split(' ');
          String shortedName = '';
```

```
        String initials = '';

        if (nameParts.length == 1) {
        shortedName = nameParts[0];
        } else if (nameParts.length == 2) {
        shortedName = nameParts.join(' ');
        } else {
        shortedName = nameParts.take(2).join(' ') + nameParts.skip(2).map((name) => '
${name[0]}.').join('');
        }

        initials =
        ((nameParts.isNotEmpty && nameParts[0].isNotEmpty ? nameParts[0][0] : '') +
(nameParts.length > 1 && nameParts[1].isNotEmpty ? nameParts[1][0] : ''))
        .toUpperCase();

        if (employee['gaji'] != null && employee['gaji'].isNotEmpty) {
        thisMonthSalary = SalaryModel.fromMap(employee['gaji'][0]);
        }
        return EmployeeModel(
        id: employee['id_karyawan'],
        name: name,
        shortedName: shortedName,
        initials: initials,
        age: employee['umur'],
        address: employee['alamat'],
        position: PositionModel.fromSupabase(employee['posisi']),
        thisMonthSalary: thisMonthSalary,
        birthPlace: employee['tempat_lahir'],
        birthDate: employee['tanggal_lahir'] != null ?
DateTime.parse(employee['tanggal_lahir']) : null,
        entryDate: employee['tanggal_masuk'] != null ?
DateTime.parse(employee['tanggal_masuk']) : null,
        phoneNumber: employee['no_telp'].toString(),
        gender: employee['gender'].toString(),
        religion: employee['agama'].toString(),
        salary: employee['gaji_pokok'] != null ? employee['gaji_pokok'].toDouble() : 0.0,
        attendanceMachineID: employee['id_mesin_absensi'] ?? 0,
        );
   }

   static List<EmployeeModel> fromSupabaseList(List<Map<String, dynamic>> employees) {
        return employees.map((employee) =>
EmployeeModel.fromSupabase(employee)).toList();
   }

   factory EmployeeModel.empty() {
        return EmployeeModel(
        id: 0,
        name: '',
        shortedName: '',
        initials: '',
        position: PositionModel.empty(),
        );
   }

   factory EmployeeModel.copyWith(EmployeeModel employee,{
        int? id,
        String? name,
        String? shortedName,
        String? initials,
        int? age,
        String? address,
        String? birthPlace,
        DateTime? birthDate,
        String? phoneNumber,
        String? gender,
```

```
        String? religion,
        int? attendanceMachineID,
        DateTime? entryDate,
        double? salary,
        PositionModel? position,
    }) {
        return EmployeeModel(
        id: id ?? employee.id,
        name: name ?? employee.name,
        shortedName: shortedName ?? employee.shortedName,
        initials: initials ?? employee.initials,
        age: age ?? employee.age,
        address: address ?? employee.address,
        birthPlace: birthPlace ?? employee.birthPlace,
        birthDate: birthDate ?? employee.birthDate,
        phoneNumber: phoneNumber ?? employee.phoneNumber,
        gender: gender ?? employee.gender,
        religion: religion ?? employee.religion,
        attendanceMachineID: attendanceMachineID ?? employee.attendanceMachineID,
        entryDate: entryDate ?? employee.entryDate,
        salary: salary ?? employee.salary,
        position: position ?? employee.position,
        );
    }
}
```

Segmen Program 4.2 *Service* Karyawan

```
class EmployeeService {
    SupabaseClient supabase = Supabase.instance.client;

    Future<List<Map<String, dynamic>>> getEmployee() async {
        try {
        final data = await supabase.from('karyawan').select('''
        *,
        posisi!inner(*)
        ''').order('nama', ascending: true);

        return data;
        } on PostgrestException catch (error) {
        throw PostgrestException(message: error.message);
        } catch (e) {
        throw Exception(e.toString());
        }
    }

    Future<List<int>> getEmployeeCount() async {
        try {
        final worker =
        await supabase.from('karyawan').select('''id_karyawan,
posisi!inner(tipe)''').eq('posisi.tipe', 1).count();
        final labor =
        await supabase.from('karyawan').select('''id_karyawan,
posisi!inner(tipe)''').eq('posisi.tipe', 2).count();

        return [worker.count, labor.count];
        } on PostgrestException catch (error) {
        throw PostgrestException(message: error.message);
        } catch (e) {
        throw Exception(e.toString());
        }
    }

    Future<void> store(EmployeeModel model) async {
        try {
        await supabase.from('karyawan').insert({
```

```
          'nama': model.name,
          'umur': model.age,
          'alamat': model.address,
          'tempat_lahir': model.birthPlace,
          'tanggal_lahir': model.birthDate?.toIso8601String(),
          'no_telp': model.phoneNumber,
          'gender': model.gender,
          'agama': model.religion,
          'id_mesin_absensi': model.attendanceMachineID,
          'tanggal_masuk': model.entryDate?.toIso8601String(),
          'gaji_pokok': model.salary,
          'id_posisi': model.position!.id,
          });
          } on PostgrestException catch (error) {
          throw PostgrestException(message: error.message);
          } catch (e) {
          throw Exception(e.toString());
          }
   }

   Future<void> update(EmployeeModel model) async {
          try {
          await supabase.from('karyawan').update({
          'nama': model.name,
          'umur': model.age,
          'alamat': model.address,
          'tempat_lahir': model.birthPlace,
          'tanggal_lahir': model.birthDate,
          'no_telp': model.phoneNumber,
          'gender': model.gender,
          'agama': model.religion,
          'id_mesin_absensi': model.attendanceMachineID,
          'tanggal_masuk': model.entryDate?.toIso8601String(),
          'gaji_pokok': model.salary,
          'id_posisi': model.position.id,
          }).eq('id_karyawan', model.id);
          } on PostgrestException catch (error) {
          throw PostgrestException(message: error.message);
          } catch (e) {
          throw Exception(e.toString());
          }
   }
}
```

Segmen Program 4.3 *ViewModel* karyawan

```
class EmployeeViewModel extends ChangeNotifier {
   final EmployeeService _employeeService;
   int workerCount = 0;
   int laborCount = 0;
   List<EmployeeModel> employee = [];
   EmployeeModel _selectedEmployee = EmployeeModel.empty();

   EmployeeViewModel({required EmployeeService employeeService}) : _employeeService =
employeeService;

   EmployeeModel get selectedEmployee => _selectedEmployee;

   void setEmployee(EmployeeModel employee) {
          _selectedEmployee = employee;
          notifyListeners();
   }
```

```
   Future<void> getEmployee() async {
          try {
          List<Map<String, dynamic>> data = await _employeeService.getEmployee();

          employee = EmployeeModel.fromSupabaseList(data);
          notifyListeners();
          } catch (e) {
          if (e is PostgrestException) {
          debugPrint('Employee Get error: ${e.message}');
          navigatorKey.currentContext!.read<ToastProvider>().showToast('Terjadi kesalahan,
mohon laporkan!', 'error');
          } else {
          debugPrint('Employee Get error: ${e.toString()}');
          navigatorKey.currentContext!.read<ToastProvider>().showToast('Terjadi kesalahan,
silahkan coba lagi!', 'error');
          }
          }
   }

   Future<void> getCount() async {
          try {
          List<int> countData = await _employeeService.getEmployeeCount();

          workerCount = countData[0];
          laborCount = countData[1];
          notifyListeners();
          } catch (e) {
          if (e is PostgrestException) {
          debugPrint('Employee Count error: ${e.message}');
          navigatorKey.currentContext!.read<ToastProvider>().showToast('Terjadi kesalahan,
mohon laporkan!', 'error');
          } else {
          debugPrint('Employee Count error: ${e.toString()}');
          navigatorKey.currentContext!.read<ToastProvider>().showToast('Terjadi kesalahan,
silahkan coba lagi!', 'error');
          }
          }
   }

   Future<void> index() async {
          await getEmployee();
          navigatorKey.currentContext!.read<NavigationService>().navigateTo(const
ManagementEmployeeScreen());
   }

   void create() {
          setEmployee(EmployeeModel.empty());
          navigatorKey.currentContext!
          .read<NavigationService>()
          .navigateTo(const ManagementEmployeeFormScreen(type: 'create'));
   }

   Future<void> store(EmployeeModel model) async {
          try {
          await _employeeService.store(model);
          navigatorKey.currentContext!.read<ToastProvider>().showToast('Data karyawan
berhasil ditambahkan', 'success');
          await index();
          } catch (e) {
          if (e is PostgrestException) {
          debugPrint('Employee Store error: ${e.message}');
          navigatorKey.currentContext!.read<ToastProvider>().showToast('Terjadi kesalahan,
mohon laporkan!', 'error');
          } else {
          debugPrint('Employee Store error: ${e.toString()}');
          navigatorKey.currentContext!.read<ToastProvider>().showToast('Terjadi kesalahan,
silahkan coba lagi!', 'error');
```

```
            }
            getEmployee();
            navigatorKey.currentContext!.read<NavigationService>().goBack();
            }
    }

    Future<void> update(EmployeeModel model) async {
            try {
            await _employeeService.update(model);
            navigatorKey.currentContext!.read<ToastProvider>().showToast('Data karyawan
berhasil diperbarui', 'success');
            await index();
            } catch (e) {
            if (e is PostgrestException) {
            debugPrint('Employee Update error: ${e.message}');
            navigatorKey.currentContext!.read<ToastProvider>().showToast('Terjadi kesalahan,
mohon laporkan!', 'error');
            } else {
            debugPrint('Employee Update error: ${e.toString()}');
            navigatorKey.currentContext!.read<ToastProvider>().showToast('Terjadi kesalahan,
silahkan coba lagi!', 'error');
            }
            getEmployee();
            navigatorKey.currentContext!.read<NavigationService>().goBack();
            }
    }

    void detail(EmployeeModel model) {
            setEmployee(model);
            navigatorKey.currentContext!.read<NavigationService>().navigateTo(const
ManagementEmployeeDetailScreen());
    }

    void edit(EmployeeModel model) {
            setEmployee(model);
            navigatorKey.currentContext!.read<NavigationService>().navigateTo(const
ManagementEmployeeFormScreen(
            type: 'edit',
            ));
    }
 }
```

4.2.2. Manajemen *User*

Segmen Program 4.4 *Model user*

```
class UserModel {
  final String id;
  final String email;
  final int level;
  final int? userChatId;
  final EmployeeModel employee;

  UserModel({
    required this.id,
    required this.email,
    required this.level,
    this.userChatId,
    required this.employee,
  });

  factory UserModel.fromSupabase(Map<String, dynamic> user) {
    int? userChatId;
    if (user['level'] == 1 && user['id_chat'] != null) {
      userChatId = user['id_chat'];
```

```
    }

    return UserModel(
      id: user['id_user'],
      email: user['email'],
      level: user['level'],
      userChatId: userChatId,
      employee: EmployeeModel.fromSupabase(user['karyawan'])
    );
  }

  static List<UserModel> fromSupabaseList(List<Map<String, dynamic>> users) {
    return users.map((user) => UserModel.fromSupabase(user)).toList();
  }
}
```

Segmen Program 4.5 *Service user*

```
class UserService {
  SupabaseClient supabase = Supabase.instance.client;

  Future<List<Map<String, dynamic>>> getUser() async {
    try {
      final data = await supabase
          .from('user')
          .select('''*, karyawan(*, posisi(*)) ''').order('email', ascending: true);
      return data;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<Map<String, dynamic>> getUserByID(String id) async {
    try {
      final data = await supabase.from('user').select('''
        *,
        karyawan(*,
          posisi(*
          )
        )
        ''').eq('id_user', id).single();

      return data;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }
}
```

Segmen Program 4.6 *ViewModel user*

```
class UserViewModel extends ChangeNotifier {
  final UserService _userService;
  final ToastProvider _toastProvider =
Provider.of<ToastProvider>(navigatorKey.currentContext!, listen: false);
  List<UserModel> _user = [];
```

```
  UserViewModel({required UserService userService}) : _userService = userService;

  get user => _user;

  Future<UserModel> getUserByID(String id) async {
    try {
      Map<String, dynamic> data = await _userService.getUserByID(id);

      return UserModel.fromSupabase(data);
    } catch (e) {
      UserModel user = UserModel.empty();
      if (e is PostgrestException) {
        debugPrint('User error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('User error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
      return user;
    }
  }

  Future<void> getUser() async {
    try {
      List<Map<String, dynamic>> data = await _userService.getUser();
      _user = UserModel.fromSupabaseList(data);

      notifyListeners();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('User error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('User error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }
}
```

4.2.3. Absensi Admin

Segmen Program 4.7 *Model* absensi *admin* & *service* absensi

```
class AttendanceAdminModel {
  EmployeeModel employee;
  AttendanceModel? attendance;
  AttendanceDetailModel? laborDetail;

  AttendanceAdminModel({
    required this.employee,
    this.attendance,
    this.laborDetail,
  });

  factory AttendanceAdminModel.fromSupabase(Map<String, dynamic> attendanceData) {
    EmployeeModel employee;
    AttendanceModel? attendance;
    AttendanceDetailModel? laborDetail;
    String name = attendanceData['nama'];
    List<String> nameParts = name.split(' ');
    String shortedName = '';
```

```
    String initials = '';

    if (nameParts.length == 1) {
      shortedName = nameParts[0];
    } else if (nameParts.length == 2) {
      shortedName = nameParts.join(' ');
    } else {
      shortedName = nameParts.take(2).join(' ') + nameParts.skip(2).map((name) => '
${name[0]}.').join('');
    }
    initials =
        ((nameParts.isNotEmpty && nameParts[0].isNotEmpty ? nameParts[0][0] : '') +
(nameParts.length > 1 && nameParts[1].isNotEmpty ? nameParts[1][0] : ''))
            .toUpperCase();

    employee = EmployeeModel(
      id: attendanceData['id_karyawan'],
      name: name,
      shortedName: shortedName,
      initials: initials,
      position: PositionModel(
        id: attendanceData['posisi']['id_posisi'],
        name: attendanceData['posisi']['nama'],
        type: attendanceData['posisi']['tipe'],
      ),
    );
    if (attendanceData['absensi'].isNotEmpty) {
      Map<String, dynamic> data = attendanceData['absensi'][0];
      DateTime? checkIn;
      DateTime? checkOut;
      int inStatus = 0;
      int outStatus = 0;
      if (data['waktu_pulang'] != null) {
        checkOut = DateTime.parse(data['waktu_pulang']);
        outStatus = 1;
      }

      if (data['waktu_masuk'] != null) {
        checkIn = DateTime.parse(data['waktu_masuk']);
        int inHour = checkIn.hour;
        if (inHour < 12) {
          inStatus = 1;
        } else {
          inStatus = 2;
        }
      }

      attendance = AttendanceModel(
        id: data['id_absensi'],
        checkIn: checkIn,
        checkOut: checkOut,
        inStatus: inStatus,
        outStatus: outStatus,
      );
    } else {
      attendance = AttendanceModel.empty();
    }

    if (attendance.id != 0 && attendanceData['absensi'][0]['absensi_detail'].isNotEmpty) {
      // laborDetail = attendance['absensi_detail'][0]
      Map<String, dynamic> laborData = attendanceData['absensi'][0]['absensi_detail'][0];

      String status = '';
      if (laborData['status'] == 1) {
        status = 'Memulai Tugas Baru';
      } else {
        status = 'Melanjutkan Tugas';
```

```
      }
      laborDetail = AttendanceDetailModel(
        id: laborData['id_detail'],
        status: laborData['status'].toInt(),
        statusName: status,
        featherType: laborData['jenis_bulu'].toInt(),
        initialQty: laborData['qty_awal'].toInt(),
        finalQty: laborData['qty_akhir'].toInt(),
        initialWeight: laborData['berat_awal'].toDouble(),
        finalWeight: laborData['berat_akhir'].toDouble(),
        minDepreciation: laborData['min_susut'].toInt(),
        performanceScore: laborData['nilai_performa'].toDouble(),
      );
    } else {
      laborDetail = AttendanceDetailModel.empty();
    }

    return AttendanceAdminModel(
      employee: employee,
      attendance: attendance,
      laborDetail: laborDetail,
    );
  }

  static List<AttendanceAdminModel> fromSupabaseList(List<Map<String, dynamic>> data) {
    return data.map((attendance) =>
AttendanceAdminModel.fromSupabase(attendance)).toList();
  }
}

class AttendanceService {
  SupabaseClient supabase = Supabase.instance.client;

  Future<List<Map<String, dynamic>>> getAdminAttendanceByDate(String date, int type) async
{
    try {
      final startTime = '$date 01:00:00';
      final endTime = '$date 23:59:59';

      final data = await Supabase.instance.client
          .from('karyawan')
          .select('''
          *,
          posisi!inner(*),
          absensi!left(*, absensi_detail(*))
          ''')
          .eq('posisi.tipe', type)
          .gte('absensi.waktu_masuk', startTime)
          .lte('absensi.waktu_masuk', endTime)
          .limit(1, referencedTable: 'absensi')
          .order('nama', ascending: true);

      return data;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<void> storeWorkerAttendance(AttendanceAdminModel model) async {
    try {
      if (model.attendance!.id !=0) {
        await supabase.from('absensi').update({
          'waktu_masuk': model.attendance!.checkIn!.toIso8601String(),
          'waktu_keluar': model.attendance!.checkOut!.toIso8601String()
        }).eq('id_absensi', model.attendance!.id).eq('id_karyawan', model.employee.id);
```

```
      }else{
        await supabase.from('absensi').insert({
          'waktu_masuk': model.attendance!.checkIn!.toIso8601String(),
          'waktu_keluar': model.attendance!.checkOut!.toIso8601String(),
          'id_karyawan': model.employee.id
        });
      }
    } on PostgrestException catch (error) {
      debugPrint('Attendance error: ${error.message}');
      throw PostgrestException(message: error.message);
    } catch (e) {
      debugPrint('Attendance error: ${e.toString()}');
      throw Exception(e.toString());
    }
  }

  Future<void> storeLaborAttendance(AttendanceAdminModel model) async {
    try {
      final List<Map<String, dynamic>> attendance =
          await supabase.from('absensi').insert({'waktu_masuk':
model.attendance!.checkIn!.toIso8601String(), 'id_karyawan': model.employee.id}).select();

      await supabase.from('absensi_detail').insert({
        'id_absensi': attendance[0]['id_absensi'],
        'jenis_bulu': model.laborDetail!.featherType,
        'status': model.laborDetail!.status,
        'qty_awal': model.laborDetail!.initialQty,
        'qty_akhir': model.laborDetail!.finalQty,
        'berat_awal': model.laborDetail!.initialWeight,
        'berat_akhir': model.laborDetail!.finalWeight,
        'min_susut': model.laborDetail!.minDepreciation,
        'nilai_performa': model.laborDetail!.performanceScore
      });
    } on PostgrestException catch (error) {
      debugPrint('Attendance error: ${error.message}');
      throw PostgrestException(message: error.message);
    } catch (e) {
      debugPrint('Attendance error: ${e.toString()}');
      throw Exception(e.toString());
    }
  }
}

class AttendanceAdminModel {
  EmployeeModel employee;
  AttendanceModel? attendance;
  AttendanceDetailModel? laborDetail;

  AttendanceAdminModel({
    required this.employee,
    this.attendance,
    this.laborDetail,
  });

  factory AttendanceAdminModel.fromSupabase(Map<String, dynamic> attendanceData) {
    EmployeeModel employee;
    AttendanceModel? attendance;
    AttendanceDetailModel? laborDetail;
    String name = attendanceData['nama'];
    List<String> nameParts = name.split(' ');
    String shortedName = '';
    String initials = '';

    if (nameParts.length == 1) {
      shortedName = nameParts[0];
    } else if (nameParts.length == 2) {
      shortedName = nameParts.join(' ');
```

```
    } else {
      shortedName = nameParts.take(2).join(' ') + nameParts.skip(2).map((name) => '
${name[0]}.').join('');
    }
    initials =
        ((nameParts.isNotEmpty && nameParts[0].isNotEmpty ? nameParts[0][0] : '') +
(nameParts.length > 1 && nameParts[1].isNotEmpty ? nameParts[1][0] : ''))
            .toUpperCase();

    employee = EmployeeModel(
      id: attendanceData['id_karyawan'],
      name: name,
      shortedName: shortedName,
      initials: initials,
      position: PositionModel(
        id: attendanceData['posisi']['id_posisi'],
        name: attendanceData['posisi']['nama'],
        type: attendanceData['posisi']['tipe'],
      ),
    );
    if (attendanceData['absensi'].isNotEmpty) {
      Map<String, dynamic> data = attendanceData['absensi'][0];
      DateTime? checkIn;
      DateTime? checkOut;
      int inStatus = 0;
      int outStatus = 0;
      if (data['waktu_pulang'] != null) {
        checkOut = DateTime.parse(data['waktu_pulang']);
        outStatus = 1;
      }

      if (data['waktu_masuk'] != null) {
        checkIn = DateTime.parse(data['waktu_masuk']);
        int inHour = checkIn.hour;
        if (inHour < 12) {
          inStatus = 1;
        } else {
          inStatus = 2;
        }
      }

      attendance = AttendanceModel(
        id: data['id_absensi'],
        checkIn: checkIn,
        checkOut: checkOut,
        inStatus: inStatus,
        outStatus: outStatus,
      );
    } else {
      attendance = AttendanceModel.empty();
    }

    if (attendance.id != 0 && attendanceData['absensi'][0]['absensi_detail'].isNotEmpty) {
      // laborDetail = attendance['absensi_detail'][0]
      Map<String, dynamic> laborData = attendanceData['absensi'][0]['absensi_detail'][0];

      String status = '';
      if (laborData['status'] == 1) {
        status = 'Memulai Tugas Baru';
      } else {
        status = 'Melanjutkan Tugas';
      }
      laborDetail = AttendanceDetailModel(
        id: laborData['id_detail'],
        status: laborData['status'].toInt(),
        statusName: status,
        featherType: laborData['jenis_bulu'].toInt(),
```

```
        initialQty: laborData['qty_awal'].toInt(),
        finalQty: laborData['qty_akhir'].toInt(),
        initialWeight: laborData['berat_awal'].toDouble(),
        finalWeight: laborData['berat_akhir'].toDouble(),
        minDepreciation: laborData['min_susut'].toInt(),
        performanceScore: laborData['nilai_performa'].toDouble(),
      );
    } else {
      laborDetail = AttendanceDetailModel.empty();
    }

    return AttendanceAdminModel(
      employee: employee,
      attendance: attendance,
      laborDetail: laborDetail,
    );
  }

  static List<AttendanceAdminModel> fromSupabaseList(List<Map<String, dynamic>> data) {
    return data.map((attendance) =>
AttendanceAdminModel.fromSupabase(attendance)).toList();
  }
}
```

Segmen Program 4.8 *Service* absensi *admin*

```
class AttendanceService {
  SupabaseClient supabase = Supabase.instance.client;

  Future<List<Map<String, dynamic>>> getAdminAttendanceByDate(String date, int type) async
{
    try {
      final startTime = '$date 01:00:00';
      final endTime = '$date 23:59:59';

      final data = await Supabase.instance.client
          .from('karyawan')
          .select('''
          *,
          posisi!inner(*),
          absensi!left(*, absensi_detail(*))
          ''')
          .eq('posisi.tipe', type)
          .gte('absensi.waktu_masuk', startTime)
          .lte('absensi.waktu_masuk', endTime)
          .limit(1, referencedTable: 'absensi')
          .order('nama', ascending: true);

      return data;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<void> storeWorkerAttendance(AttendanceAdminModel model) async {
    try {
      if (model.attendance!.id !=0) {
        await supabase.from('absensi').update({
          'waktu_masuk': model.attendance!.checkIn!.toIso8601String(),
          'waktu_keluar': model.attendance!.checkOut!.toIso8601String()
        }).eq('id_absensi', model.attendance!.id).eq('id_karyawan', model.employee.id);
      }else{
        await supabase.from('absensi').insert({
```

```
          'waktu_masuk': model.attendance!.checkIn!.toIso8601String(),
          'waktu_keluar': model.attendance!.checkOut!.toIso8601String(),
          'id_karyawan': model.employee.id
        });
      }
    } on PostgrestException catch (error) {
      debugPrint('Attendance error: ${error.message}');
      throw PostgrestException(message: error.message);
    } catch (e) {
      debugPrint('Attendance error: ${e.toString()}');
      throw Exception(e.toString());
    }
  }

  Future<void> storeLaborAttendance(AttendanceAdminModel model) async {
    try {
      final List<Map<String, dynamic>> attendance =
          await supabase.from('absensi').insert({'waktu_masuk':
model.attendance!.checkIn!.toIso8601String(), 'id_karyawan': model.employee.id}).select();

      await supabase.from('absensi_detail').insert({
        'id_absensi': attendance[0]['id_absensi'],
        'jenis_bulu': model.laborDetail!.featherType,
        'status': model.laborDetail!.status,
        'qty_awal': model.laborDetail!.initialQty,
        'qty_akhir': model.laborDetail!.finalQty,
        'berat_awal': model.laborDetail!.initialWeight,
        'berat_akhir': model.laborDetail!.finalWeight,
        'min_susut': model.laborDetail!.minDepreciation,
        'nilai_performa': model.laborDetail!.performanceScore
      });
    } on PostgrestException catch (error) {
      debugPrint('Attendance error: ${error.message}');
      throw PostgrestException(message: error.message);
    } catch (e) {
      debugPrint('Attendance error: ${e.toString()}');
      throw Exception(e.toString());
    }
  }
}
```

Segmen Program 4.9 *ViewModel Absensi*

```
class AttendanceViewModel extends ChangeNotifier {
  final AttendanceService _attendanceService;
  final ToastProvider _toastProvider =
Provider.of<ToastProvider>(navigatorKey.currentContext!, listen: false);
  final DateProvider _dateProvider =
Provider.of<DateProvider>(navigatorKey.currentContext!, listen: false);
  final ConfigurationProvider _configurationProvider =
Provider.of<ConfigurationProvider>(navigatorKey.currentContext!, listen: false);
  int workerCount = 0;
  int laborCount = 0;
  List<AttendanceAdminModel> adminAttendance = [];
  AttendanceAdminModel _selectedAtt = AttendanceAdminModel.empty();
  DateTime _selectedDateAttAdmin = DateTime.now();

  AttendanceViewModel({required AttendanceService attendanceService}) : _attendanceService
= attendanceService;

  get attendanceAdmin => adminAttendance;
  AttendanceAdminModel get selectedAtt => _selectedAtt;
  get selectedAdminDate => _selectedDateAttAdmin;
```

```
  void setAttendanceAdmin(AttendanceAdminModel model) {
    _selectedAtt = model;
    notifyListeners();
  }

  void setAdminDate(DateTime date) {
    _selectedDateAttAdmin = date;
    notifyListeners();
  }


  Future<void> getAdminAttendance(int type) async {
    try {
      String date = '';
      List<Map<String, dynamic>>? data;
      if (type == 1) {
        date = _dateProvider.attendanceWorkerDateString;
        data = await _attendanceService.getAdminAttendanceByDate(date, type);
      } else if (type == 2) {
        date = _dateProvider.attendanceLaborDateString;
        data = await _attendanceService.getAdminAttendanceByDate(date, type);
      }
      adminAttendance = AttendanceAdminModel.fromSupabaseList(data!);

      notifyListeners();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Get Attendance error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Get Attendance error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  Future<void> storeWorker(String startTime, String endTime) async {
    try {
      DateTime checkIn = DateTime(
        _selectedDateAttAdmin.year,
        _selectedDateAttAdmin.month,
        _selectedDateAttAdmin.day,
        int.parse(startTime.split(':')[0]),
        int.parse(startTime.split(':')[1]),
      );

      DateTime checkOut = DateTime(
        _selectedDateAttAdmin.year,
        _selectedDateAttAdmin.month,
        _selectedDateAttAdmin.day,
        int.parse(endTime.split(':')[0]),
        int.parse(endTime.split(':')[1]),
      );
      final attendanceModel = AttendanceAdminModel(
          employee: _selectedAtt.employee,
          attendance: AttendanceModel(id: selectedAtt.attendance!.id, checkIn: checkIn,
checkOut: checkOut),
          laborDetail: AttendanceDetailModel.empty());

      await _attendanceService.storeWorkerAttendance(attendanceModel);

      getAdminAttendance(1);
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Store Attendance Worker error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
```

```
        debugPrint('Store Attendance Worker error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  Future<void> storeLabor(
      int attendanceStatus, int featherType, int initialQty, int finalQty, double
initialWeight, double finalWeight, int minDeprecation) async {
    try {
      double minimunWeightLoss = initialWeight * (minDeprecation / 100);
      double weightLoss = initialWeight - finalWeight;
      if (weightLoss < minimunWeightLoss) weightLoss = minimunWeightLoss;
      double normalizedLoss = weightLoss / (initialWeight - minimunWeightLoss);
      double score = 100 - (normalizedLoss * 100);

      final attendanceModel = AttendanceAdminModel(
          employee: _selectedAtt.employee,
          attendance: AttendanceModel(id: 0, checkIn: _selectedDateAttAdmin),
          laborDetail: AttendanceDetailModel(
            id: 0,
            status: attendanceStatus,
            featherType: featherType,
            initialQty: initialQty,
            finalQty: finalQty,
            initialWeight: initialWeight,
            finalWeight: finalWeight,
            minDepreciation: minDeprecation,
            performanceScore: score,
          ));
      await _attendanceService.storeLaborAttendance(attendanceModel);

      getAdminAttendance(2);
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Store Attendance Labor error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Store Attendance Labor error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }
}
```

4.2.4. Absensi *User*

Segmen Program 4.10 *Model* absensi

```
class AttendanceUserModel {
  final DateTime date;
  final AttendanceModel? attendance;
  final AttendanceDetailModel? laborDetail;

  AttendanceUserModel({
    required this.date,
    // this.attendance,
    // this.laborDetail,
    required this.attendance,
    required this.laborDetail,
```

```
  });

  factory AttendanceUserModel.fromSupabase(List<Map<String, dynamic>> attendances, DateTime
date) {
    AttendanceModel? attendance;
    AttendanceDetailModel? laborDetail;
    if (attendances[0]['absensi'].isNotEmpty) {
      List<dynamic> attendanceData = attendances[0]['absensi'];

      for (var i = 0; i < attendanceData.length; i++) {
        Map<String, dynamic> data = attendanceData[i];
        // Map<String, dynamic>? laborDetail;
        DateTime checkIn = DateTime.parse(data['waktu_masuk']);
        DateTime? checkOut;

        if (data['waktu_pulang'] != null) {
          checkOut = DateTime.parse(data['waktu_pulang']);
        }

        if (checkIn.year == date.year && checkIn.month == date.month && checkIn.day ==
date.day) {
          int inHour = checkIn.hour;
          int inStatus = 0;
          if (inHour < 12) {
            inStatus = 1;
          } else {
            inStatus = 2;
          }

          int outStatus = 0;
          if (checkOut != null) {
            outStatus = 1;
          }
          attendance = AttendanceModel(
            id: data['id_absensi'],
            checkIn: checkIn,
            checkOut: checkOut,
            inStatus: inStatus,
            outStatus: outStatus,
          );

          if (data['absensi_detail'].isNotEmpty) {
            // laborDetail = attendance['absensi_detail'][0]
            Map<String, dynamic> laborData = attendanceData[0]['absensi_detail'][0];

            String status = '';
            if (data['absensi_detail'][0]['status'] == 1) {
              status = 'Memulai Tugas Baru';
            } else {
              status = 'Melanjutkan Tugas';
            }


            laborDetail = AttendanceDetailModel(
              id: laborData['id_detail'],
              status: laborData['status'].toInt(),
              statusName: status,
              featherType: laborData['jenis_bulu'].toInt(),
              initialQty: laborData['qty_awal'].toInt(),
              finalQty: laborData['qty_akhir'].toInt(),
              initialWeight: laborData['berat_awal'].toDouble(),
              finalWeight: laborData['berat_akhir'].toDouble(),
              minDepreciation: laborData['min_susut'].toInt(),
              performanceScore: laborData['nilai_performa'].toDouble(),
            );
          }
```

```
          return AttendanceUserModel(
            date: date,
            attendance: attendance,
            laborDetail: laborDetail,
          );
        }
      }
    }
    return AttendanceUserModel(
      date: date,
      attendance: AttendanceModel.empty(),
      laborDetail: AttendanceDetailModel.empty(),
    );
  }

  static List<AttendanceUserModel> fromSupabaseList(List<Map<String, dynamic>> attendances,
List<DateTime> dates) {
    return dates.map((date) {
      return AttendanceUserModel.fromSupabase(attendances, date);
    }).toList();
  }
}
```

Segmen Program 4.11 *Service* absensi

```
class AttendanceService {
  SupabaseClient supabase = Supabase.instance.client;


   Future<List<Map<String, dynamic>>> getUserAttendance(int employeeID, String startDate,
String endDate) async {
    try {
      final startTime = '$startDate 01:00:00';
      final endTime = '$endDate 23:59:59';

      final data = await supabase.from('karyawan').select('''
          *,
          absensi!left(*, absensi_detail!left(*))
          ''').gte('absensi.waktu_masuk', startTime).lte('absensi.waktu_masuk',
endTime).eq('id_karyawan', employeeID);

      return data;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }
}
```

Segmen Program 4.12 *ViewModel* absensi

```
class AttendanceViewModel extends ChangeNotifier {
  final AttendanceService _attendanceService;
  final ToastProvider _toastProvider =
Provider.of<ToastProvider>(navigatorKey.currentContext!, listen: false);
  final DateProvider _dateProvider =
Provider.of<DateProvider>(navigatorKey.currentContext!, listen: false);
  final ConfigurationProvider _configurationProvider =
Provider.of<ConfigurationProvider>(navigatorKey.currentContext!, listen: false);
  List<AttendanceUserModel> userAttendance = [];

  AttendanceViewModel({required AttendanceService attendanceService}) : _attendanceService
= attendanceService;
```

```
  get attendanceUser => userAttendance;


  Future<void> getUserAttendance() async {
    try {
      String startDate = _dateProvider.attendanceUserDateStartString;
      String endDate = _dateProvider.attendanceUserDateEndString;
      int employeeID = _configurationProvider.user.employee.id;
      List<Map<String, dynamic>> data = await
_attendanceService.getUserAttendance(employeeID, startDate, endDate);

      List<DateTime> dateRange = generateDateRange(_dateProvider.attendanceUserDateStart,
_dateProvider.attendanceUserDateEnd);

      userAttendance = AttendanceUserModel.fromSupabaseList(data, dateRange);
      notifyListeners();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Attendance error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Attendance error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  List<DateTime> generateDateRange(DateTime startDate, DateTime endDate) {
    List<DateTime> dateRange = [];
    for (int i = 0; i <= endDate.difference(startDate).inDays; i++) {
      DateTime currentDate = startDate.add(Duration(days: i));
      if (currentDate.weekday != DateTime.sunday) {
        dateRange.add(currentDate);
      }
    }
    return dateRange;
  }
}
```

4.2.5. Perizinan

Segmen Program 4.13 *Model* izin

```
class RequestModel {
  final int id;
  final int type;
  final int status;
  final DateTime? startDateTime;
  final DateTime? endDateTime;
  final DateTime? approvalTime;
  final DateTime? rejectTime;
  final String? reason;
  final String? comment;
  final String? file;
  final EmployeeModel requester;
  final EmployeeModel? approver;
  final EmployeeModel? rejecter;

  RequestModel({
```

```
    required this.id,
    required this.type,
    required this.status,
    this.startDateTime,
    this.endDateTime,
    this.approvalTime,
    this.rejectTime,
    this.reason,
    this.comment,
    this.file,
    required this.requester,
    this.approver,
    this.rejecter,
  });

  static List<RequestModel> fromSupabaseList(List<Map<String, dynamic>> requests) {
    return requests.map((request) {
      bool haveApprover = request['approver'] != null;
      bool haveRejecter = request['penolak'] != null;
      EmployeeModel approver;
      EmployeeModel rejecter;

      if (haveApprover){
        approver = EmployeeModel.fromSupabase(request['approver']);
      } else {
        approver = EmployeeModel.empty();
      }

      if (haveRejecter){
        rejecter = EmployeeModel.fromSupabase(request['penolak']);
      } else {
        rejecter = EmployeeModel.empty();
      }

      return RequestModel(
        id: request['id_izin'],
        type: request['jenis'],
        status: request['status'],
        startDateTime: request['waktu_mulai'] == null ? null :
DateTime.tryParse(request['waktu_mulai']),
        endDateTime: request['waktu_akhir'] == null ? null :
DateTime.tryParse(request['waktu_akhir']),
        approvalTime: request['waktu_approve'] == null ? null :
DateTime.tryParse(request['waktu_approve']),
        rejectTime: request['waktu_tolak'] == null ? null :
DateTime.tryParse(request['waktu_tolak']),
        requester: EmployeeModel.fromSupabase(request['karyawan']),
        approver: approver,
        rejecter: rejecter,
        reason: request['alasan'] ?? '',
        comment: haveRejecter || haveApprover ? request['komentar'] : '',
        file: request['file'] ?? '',
      );
    }).toList();
  }
}
```

Segmen Program 4.14 *Service* izin

```
class RequestService {
  SupabaseClient supabase = Supabase.instance.client;

  Future<List<Map<String, dynamic>>> getDashboardRequest(int userLevel, {int? employeeID})
async {
```

```
  try {
    var query = supabase.from('izin').select('''
      id_izin,
      jenis,
      status,
      waktu_mulai,
      waktu_akhir,
      alasan,
      komentar,
      waktu_approve,
      waktu_tolak,
      file,
      karyawan:id_karyawan(
        *,
        posisi(
          *
        )
      ),
      approver:id_approver(
        *,
        posisi(
          *
        )
      ),
      penolak:id_penolak(
        *,
        posisi(
          *
        )
      )
      ''');

    if (userLevel > 1) query = query.eq('status', 0);

    if (employeeID != null) query = query.eq('id_karyawan', employeeID);

    final data = await query.order('id_izin', ascending: false).limit(3);

    return data;
  } on PostgrestException catch (error) {
    throw PostgrestException(message: error.message);
  } catch (e) {
    throw Exception(e.toString());
  }
}

Future<List<Map<String, dynamic>>> getListRequest({int? employeeID}) async {
  try {
    final query = supabase.from('izin').select('''
      id_izin,
      jenis,
      status,
      waktu_mulai,
      waktu_akhir,
      alasan,
      komentar,
      waktu_approve,
      waktu_tolak,
      file,
      karyawan:id_karyawan(
        *,
        posisi(
          *
        )
      ),
      approver:id_approver(
        *,
```

```
          posisi(
            *
          )
        ),
        penolak:id_penolak(
          *,
          posisi(
            *
          )
        )
        ''');

      if (employeeID != null) query.eq('id_karyawan', employeeID);

      final data = await query.order('id_izin', ascending: false);

      return data;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<void> store(int employeeID, int type, String reason, {File? file, String?
startTime, String? endTime}) async {
    try {
      String path = '';
      if (file != null) {
        final String fileName = file.path.split('/').last;
        path = await supabase.storage.from('RequestFile').upload(
              '/$employeeID/$fileName',
              file,
              fileOptions: const FileOptions(cacheControl: '3600', upsert: false),
            );
      }
      await supabase.from('izin').insert({
        'id_karyawan': employeeID,
        'jenis': type,
        'waktu_mulai': startTime,
        'waktu_akhir': endTime,
        'alasan': reason,
        'file': path,
        'update_oleh': employeeID,
      });
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<void> update(int id, int employeeID, int type, String reason, {File? file, String?
startTime, String? endTime}) async {
    try {
      String path = '';
      if (file != null) {
        final String fileName = file.path.split('/').last;
        path = await supabase.storage.from('RequestFile').upload(
              '/$employeeID/$fileName',
              file,
              fileOptions: const FileOptions(cacheControl: '3600', upsert: false),
            );

        await supabase.from('izin').update({
          'id_karyawan': employeeID,
          'jenis': type,
```

```
          'waktu_mulai': startTime,
          'waktu_akhir': endTime,
          'alasan': reason,
          'file': path,
          'update_oleh': employeeID,
        }).eq('id_izin', id);
      } else {
        await supabase.from('izin').update({
          'id_karyawan': employeeID,
          'jenis': type,
          'waktu_mulai': startTime,
          'waktu_akhir': endTime,
          'alasan': reason,
          'update_oleh': employeeID,
        }).eq('id_izin', id);
      }
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<void> approve(int id, int employeeID, String? comment) async {
    try {
      await supabase.from('izin').update({
        'id_approver': employeeID,
        'komentar': comment,
        'status': 1,
        'waktu_approve': DateTime.now().toIso8601String(),
        'update_oleh': employeeID,
      }).eq('id_izin', id);
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<void> reject(int id, int employeeID, String comment) async {
    try {
      await supabase.from('izin').update({
        'id_penolak': employeeID,
        'komentar': comment,
        'status': 2,
        'waktu_tolak': DateTime.now().toIso8601String(),
        'update_oleh': employeeID,
      }).eq('id_izin', id);
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }
}
```

Segmen Program 4.15 *ViewModel* izin

```
class RequestViewModel extends ChangeNotifier {
  final RequestService _requestService;
  final ToastProvider _toastProvider =
Provider.of<ToastProvider>(navigatorKey.currentContext!, listen: false);
  final FileProvider _fileProvider =
Provider.of<FileProvider>(navigatorKey.currentContext!, listen: false);
```

```
  final ConfigurationProvider _configurationProvider =
Provider.of<ConfigurationProvider>(navigatorKey.currentContext!, listen: false);
  List<RequestModel> _request = [];
  List<RequestModel> _requestDashboard = [];
  List<int> _filterCategory = [];
  String filterStatus = 'Pending';
  DateTimeRange? filterDate;

  RequestViewModel({required RequestService requestService}) : _requestService =
requestService;

  get request {
    List<RequestModel> requestFiltered = _request;

    if (filterStatus == '') {
      requestFiltered = requestFiltered
          .where((element) => element.approver!.id == 0 || element.rejecter!.id == 0 ||
element.approver!.id != 0 || element.rejecter!.id != 0)
          .toList();
    } else {
      if (filterStatus == 'Pending') {
        requestFiltered = requestFiltered.where((element) => element.approver!.id == 0 &&
element.rejecter!.id == 0).toList();
      }

      if (filterStatus == 'Approved') {
        requestFiltered = requestFiltered.where((element) => element.approver!.id !=
0).toList();
      }

      if (filterStatus == 'Rejected') {
        requestFiltered = requestFiltered.where((element) => element.rejecter!.id !=
0).toList();
      }
    }

    if (_filterCategory.isNotEmpty) {
      requestFiltered = requestFiltered.where((element) =>
_filterCategory.contains(element.type)).toList();
    }

    if (filterDate != null) {
      requestFiltered = requestFiltered.where((element) {
        return element.startDateTime!.isAfter(filterDate!.start) &&
element.startDateTime!.isBefore(filterDate!.end);
      }).toList();
    }

    return requestFiltered;
  }

  get requestDashboard => _requestDashboard;

  Future<void> getDashboardRequest() async {
    try {
      List<Map<String, dynamic>> data;
      data = await _requestService.getDashboardRequest(_configurationProvider.user.level,
          employeeID: _configurationProvider.isAdmin ? null :
_configurationProvider.user.employee.id);

      _requestDashboard = RequestModel.fromSupabaseList(data);
      notifyListeners();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Request get dashboard error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
```

```
        debugPrint('Request get dashboard error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  Future<void> getRequest() async {
    try {
      List<Map<String, dynamic>> data;
      data = await _requestService.getListRequest(employeeID:
_configurationProvider.isAdmin ? null : _configurationProvider.user.employee.id);
      _request = RequestModel.fromSupabaseList(data);
      notifyListeners();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Request get list error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Request get list error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  Future<void> store(int type, String reason, {String? fileName, String? startTime, String?
endTime, String? startDate, String? endDate}) async {
    try {
      final int employeeID = _configurationProvider.user.employee.id;
      File? file;
      String? startDateTime;
      String? endDateTime;

      if ((fileName != null && fileName != '') && fileName == _fileProvider.fileName) {
        file = _fileProvider.file;
      }
      if (type == 1 || type == 2) {
        if (startTime != null && startTime != '') {
          startDateTime = DateFormat('dd/MM/yyyy HH:mm').parse('$startDate
$startTime').toIso8601String();
        }
      } else {
        startDateTime = DateFormat('dd/MM/yyyy HH:mm').parse('$startDate
$startTime').toIso8601String();
        endDateTime = DateFormat('dd/MM/yyyy HH:mm').parse('$endDate
$endTime').toIso8601String();
      }

      await _requestService.store(employeeID, type, reason, file: file, startTime:
startDateTime, endTime: endDateTime);
      _toastProvider.showToast('Permintaan berhasil disimpan!', 'success');


    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Request store error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Request store error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  Future<void> update(RequestModel model, int type, String reason,
      {String? fileName, String? startTime, String? endTime, String? startDate, String?
endDate}) async {
    try {
```

```
      final int employeeID = _configurationProvider.user.employee.id;
      File? file;
      String? startDateTime;
      String? endDateTime;

      if ((fileName != null && fileName != '' && fileName == model.file?.split('/').last)
&& fileName == _fileProvider.fileName) {
        file = _fileProvider.file;
      }
      if (type == 1 || type == 2) {
        if (startTime != null && startTime != '') {
          startDateTime = DateFormat('dd/MM/yyyy HH:mm').parse('$startDate
$startTime').toIso8601String();
        }
      } else {
        startDateTime = DateFormat('dd/MM/yyyy HH:mm').parse('$startDate
$startTime').toIso8601String();
        endDateTime = DateFormat('dd/MM/yyyy HH:mm').parse('$endDate
$endTime').toIso8601String();
      }

      await _requestService.update(model.id, employeeID, type, reason, file: file,
startTime: startDateTime, endTime: endDateTime);
      _toastProvider.showToast('Permintaan berhasil diupdate!', 'success');

    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Request update error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Request update error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  Future<void> response(String type, int id, String? comment) async {
    try {
      final int employeeID = _configurationProvider.user.employee.id;
      if (type == 'approve') {
        await _requestService.approve(id, employeeID, comment);
        _toastProvider.showToast('Permintaan berhasil disetujui!', 'success');
      } else {
        await _requestService.reject(id, employeeID, comment!);
        _toastProvider.showToast('Permintaan berhasil ditolak!', 'success');
      }
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Request response error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Request response error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  void addFilter({List<int>? selectedCategory, String? status, DateTimeRange? date}) {
    if (selectedCategory != null) {
      _filterCategory = selectedCategory;
    } else {
      _filterCategory = [];
    }
    if (status != null) {
      filterStatus = status;
    } else {
      filterStatus = 'Pending';
```

```
    }
    if (date != null) {
      filterDate = date;
    } else {
      filterDate = null;
    }
    notifyListeners();
  }
}
```

4.2.6. Pengumuman

Segmen Program 4.16 *Model* pengumuman

```
class AnnouncementModel {
  final int id;
  final String title;
  final String content;
  final DateTime? postDate;
  final int duration;
  final bool isSend;
  final int status;
  final EmployeeModel employee;
  final AnnouncementCategoryModel category;

  AnnouncementModel({
    required this.id,
    required this.title,
    required this.content,
    this.postDate,
    required this.duration,
    this.isSend = false,
    this.status = 0,
    required this.employee,
    required this.category,
  });

  factory AnnouncementModel.fromSupabase(Map<String, dynamic> announcement) {
    return AnnouncementModel(
      id: announcement['id_pengumuman'],
      title: announcement['judul'],
      content: announcement['isi'],
      postDate: announcement['waktu_kirim'] != null ?
DateTime.parse(announcement['waktu_kirim']) : null,
      duration: int.parse(announcement['durasi']),
      isSend: announcement['sudah_kirim'],
      status: announcement['status'],
      employee: EmployeeModel.fromSupabase(announcement['karyawan']),
      category:
AnnouncementCategoryModel.fromSupabase(announcement['pengumuman_kategori']),
    );
  }

  static List<AnnouncementModel> fromSupabaseList(List<Map<String, dynamic>> announcements)
{
    return announcements.map((announcement) =>
AnnouncementModel.fromSupabase(announcement)).toList();
  }
}
```

Segmen Program 4.17 *Service* pengumuman

```
class AnnouncementService {
  SupabaseClient supabase = Supabase.instance.client;

  Future<List<Map<String,dynamic>>> getDashboardAnnouncement() async {
    try {
      final data = await supabase.from('pengumuman').select('''
        *,
        pengumuman_kategori!inner(*)
      ''').order('tanggal_post', ascending: false).limit(2);

      return data;
    } on PostgrestException catch (error) {
      debugPrint('Announcement error: ${error.message}');
      throw PostgrestException(message: error.message);
    } catch (e) {
      debugPrint('Announcement error: ${e.toString()}');
      throw Exception(e.toString());
    }
  }

  Future<List<Map<String,dynamic>>> getAnnouncement() async {
    try {
      final data = await supabase.from('pengumuman').select('''
        *,
        pengumuman_kategori!inner(*),
        karyawan!inner(*, posisi!inner(*))
      ''').order('waktu_kirim', ascending: false);

      return data;
    } on PostgrestException catch (error) {
      debugPrint('Announcement error: ${error.message}');
      throw PostgrestException(message: error.message);
    } catch (e) {
      debugPrint('Announcement error: ${e.toString()}');
      throw Exception(e.toString());
    }
  }

  Future<List<Map<String,dynamic>>> getAnnouncementByCategory(List<int> categoryID) async {
    try {
      final data = await supabase.from('pengumuman').select('''
        *,
        pengumuman_kategori!inner(*)
      ''').inFilter('id_kategori', categoryID).order('tanggal_post', ascending: false);

      return data;
    } on PostgrestException catch (error) {
      debugPrint('Announcement error: ${error.message}');
      throw PostgrestException(message: error.message);
    } catch (e) {
      debugPrint('Announcement error: ${e.toString()}');
      throw Exception(e.toString());
    }
  }

  Future<List<Map<String,dynamic>>> getAnnouncementCategory() async {
    try {
      final data = await
supabase.from('pengumuman_kategori').select('*').order('id_kategori', ascending: true);

      return data;
    } on PostgrestException catch (error) {
      debugPrint('Announcement error: ${error.message}');
```

```
      throw PostgrestException(message: error.message);
    } catch (e) {
      debugPrint('Announcement error: ${e.toString()}');
      throw Exception(e.toString());
    }
  }

  Future<void> storeAnnouncement(
      int categoryId, String title, String content, DateTime dateTime, int duration, bool
isPosted) async {
    try {
      await supabase.from('pengumuman').insert({
        'id_kategori': categoryId,
        'judul': title,
        'isi': content,
        'waktu_kirim': dateTime.toIso8601String(),
        'durasi' : duration,
        'sudah_kirim': isPosted,
        'id_pembuat': Provider.of<ConfigurationProvider>(navigatorKey.currentContext!,
listen: false).user.employee.id,
      });
    } on PostgrestException catch (error) {
      debugPrint('Announcement error: ${error.message}');
      throw PostgrestException(message: error.message);
    } catch (e) {
      debugPrint('Announcement error: ${e.toString()}');
      throw Exception(e.toString());
    }
  }

  Future<void> storeAnnouncementCategory(String title, String color) async {
    try {
      await supabase.from('AnnouncementCategory').insert({
        'name': title,
        'color': color,
      });
    } on PostgrestException catch (error) {
      debugPrint('Announcement error: ${error.message}');
      throw PostgrestException(message: error.message);
    } catch (e) {
      debugPrint('Announcement error: ${e.toString()}');
      throw Exception(e.toString());
    }
  }

  Future<void> updateAnnouncementCategory(int id, String title, String color) async {
    try {
      await supabase.from('AnnouncementCategory').update({
        'name': title,
        'color': color,
      }).eq('category_id', id);
    } on PostgrestException catch (error) {
      debugPrint('Announcement error: ${error.message}');
      throw PostgrestException(message: error.message);
    } catch (e) {
      debugPrint('Announcement error: ${e.toString()}');
      throw Exception(e.toString());
    }
  }
}
```

Segmen Program 4.18 *ViewModel* pengumuman

```
class AnnouncementViewModel extends ChangeNotifier {
  final AnnouncementService _announcementService;
```

```
  final ToastProvider _toastProvider =
Provider.of<ToastProvider>(navigatorKey.currentContext!, listen: false);
  List<AnnouncementModel> _announcement = [];
  List<AnnouncementModel> _announcementDashboard = [];
  List<AnnouncementCategoryModel> _announcementCategory = [];
  List<int> selectedCategory = [];

  AnnouncementViewModel({required AnnouncementService announcementService}) :
_announcementService = announcementService;

  get announcement => _announcement;
  get announcementDashboard => _announcementDashboard;
  get announcementCategory => _announcementCategory;

  Future<void> getDashboardAnnouncement() async {
    try {
      List<Map<String, dynamic>> data = await
_announcementService.getDashboardAnnouncement();
      _announcementDashboard = AnnouncementModel.fromSupabaseList(data);

      notifyListeners();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Announcement error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Announcement error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  Future<void> getAnnouncement({List<int> categoryID = const []}) async {
    try {
      List<Map<String, dynamic>> data = [];
      if (categoryID.isEmpty) {
        data = await _announcementService.getAnnouncement();
      } else {
        data = await _announcementService.getAnnouncementByCategory(categoryID);
      }
      _announcement = AnnouncementModel.fromSupabaseList(data);

      notifyListeners();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Announcement error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Announcement error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  Future<void> getAnnouncementCategory() async {
    try {
      List<Map<String, dynamic>> data = await
_announcementService.getAnnouncementCategory();
      _announcementCategory = AnnouncementCategoryModel.fromSupabaseList(data);

      notifyListeners();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Announcement error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Announcement error: ${e.toString()}');
```

```
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  Future<void> store(int categoryId, String title, String content, String date, String
time, int duration, bool isPosted) async {
    try {
      DateTime tempDate = DateFormat('dd-MM-yyyy').parse(date);
      DateTime tempTime = DateFormat('HH:mm').parse(time);
      DateTime postDate = DateTime(tempDate.year, tempDate.month, tempDate.day,
tempTime.hour, tempTime.minute);

      await _announcementService.storeAnnouncement(categoryId, title, content, postDate,
duration, isPosted);
      _toastProvider.showToast('Berhasil menambahkan pengumuman!', 'success');

      getAnnouncement();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Announcement error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Announcement error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  Future<void> storeAnnouncementCategory(String title, String color) async {
    try {
      await _announcementService.storeAnnouncementCategory(title, color);
      _toastProvider.showToast('Berhasil menambahkan kategori pengumuman!', 'success');

      getAnnouncementCategory();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Announcement error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Announcement error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }

  Future<void> updateAnnouncementCategory(AnnouncementCategoryModel model, String title,
String color) async {
    try {
      await _announcementService.updateAnnouncementCategory(model.id, title, color);
      _toastProvider.showToast('Berhasil mengubah kategori pengumuman!', 'success');

      getAnnouncementCategory();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Announcement error: ${e.message}');
        _toastProvider.showToast('Terjadi kesalahan, mohon laporkan!', 'error');
      } else {
        debugPrint('Announcement error: ${e.toString()}');
        _toastProvider.showToast('Terjadi kesalahan, silahkan coba lagi!', 'error');
      }
    }
  }
}
```

4.2.7. Gaji

Segmen Program 4.19 *Model* gaji

```
class SalaryModel {
  int id;
  int employeeId;
  double monthlySalary;
  double totalSalary;
  double totalBPJS;
  double totalOvertime;
  double totalDeduction;
  double totalBonus;
  String period;
  int totalAttendance;
  int totalWorkingDay;
  bool isSaved;

  SalaryModel({
    required this.id,
    required this.employeeId,
    required this.monthlySalary,
    required this.totalSalary,
    required this.totalBPJS,
    required this.totalOvertime,
    required this.totalDeduction,
    required this.totalBonus,
    required this.period,
    required this.totalAttendance,
    required this.totalWorkingDay,
    this.isSaved = false,
  });

  factory SalaryModel.fromMap(Map<String, dynamic> salary) {
    return SalaryModel(
      id: salary['id_gaji'],
      employeeId: salary['id_karyawan'],
      monthlySalary: salary['gaji'] ?? 0,
      totalSalary: salary['total_gaji'] ?? 0,
      totalBPJS: salary['bpjs'] ?? 0,
      totalOvertime: salary['lembur'] ?? 0,
      totalDeduction: salary['potongan'] ?? 0,
      totalBonus: salary['tunjangan'] ?? 0,
      period: salary['periode'],
      totalAttendance: salary['jumlah_kehadiran'] ?? 0,
      totalWorkingDay: salary['jumlah_hari_kerja']  ?? 0,
      isSaved: true,
    );
  }

  factory SalaryModel.fromFunction(List<dynamic> salary, String period, int employeeId) {
    return SalaryModel(
      id: 0,
      employeeId: employeeId,
      monthlySalary: salary[0]['monthly_wage'].toDouble(),
      totalSalary: salary[0]['total_salary'].toDouble(),
      totalBPJS: salary[0]['total_bpjs'].toDouble(),
      totalOvertime: salary[0]['total_overtime'].toDouble(),
      totalDeduction: salary[0]['total_cut'].toDouble(),
      totalBonus: 0,
      period: period,
      totalAttendance: salary[0]['total_attendance'],
      totalWorkingDay: salary[0]['total_working'],
      isSaved: false,
    );
```

```
  }

  static List<SalaryModel> fromMapList(List<Map<String, dynamic>> salaries) {
    return salaries.map((salary) => SalaryModel.fromMap(salary)).toList();
  }
}
```

Segmen Program 4.20 *Service* gaji

```
class SalaryService {
  SupabaseClient supabase = Supabase.instance.client;

  Future<Map<String, dynamic>> getEmployeeSalary(int employeeID, String period) async {
    try {
      final data = await supabase.from('gaji').select('*').eq('id_karyawan',
employeeID).eq('periode', period);
      if (data.isEmpty) {
        return {};
      } else {
        return data[0];
      }
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<List<dynamic>> getWorkerSalary(int employeeID, String startDate, String endDate)
async {
    try {
      final data = await supabase.rpc('calculate_salary', params: {'emp_id': employeeID,
'start_date': startDate, 'end_date': endDate});
      return data;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<List<dynamic>> getLaborSalary(int employeeID, String startDate, String endDate)
async {
    try {
      final data = await supabase.rpc('calculate_salary_labor', params: {'emp_id':
employeeID, 'start_date': startDate, 'end_date': endDate});
      return data;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }


  Future<void> store(EmployeeModel employee, SalaryModel salary) async {
    try {
      await supabase.from('gaji').upsert([
        {
          'id_karyawan': employee.id,
          'periode': salary.period,
          'gaji': salary.monthlySalary,
          'tunjangan': salary.totalBonus,
```

```
          'lembur': salary.totalOvertime,
          'bpjs': salary.totalBPJS,
          'potongan': salary.totalDeduction,
          'total_gaji': salary.totalSalary,
          'keterangan': salary.note,
          'jumlah_kehadiran': salary.totalAttendance,
          'jumlah_hari_kerja': salary.totalWorkingDay
        }
      ]);
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<void> update(EmployeeModel employee, SalaryModel salary) async {
    try {
      await supabase.from('gaji').update({
        'id_karyawan': employee.id,
        'periode': salary.period,
        'gaji': salary.monthlySalary,
        'tunjangan': salary.totalBonus,
        'lembur': salary.totalOvertime,
        'bpjs': salary.totalBPJS,
        'potongan': salary.totalDeduction,
        'total_gaji': salary.totalSalary,
        'keterangan': salary.note,
        'jumlah_kehadiran': salary.totalAttendance,
        'jumlah_hari_kerja': salary.totalWorkingDay
      }).eq('id_gaji', salary.id);
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }
}
```

Segmen Program 4.21 *Viewmodel* gaji

```
class SalaryViewModel extends ChangeNotifier {
  final SalaryService _salaryService;
  SalaryModel _selectedSalary = SalaryModel.empty();
  EmployeeModel selectedEmployee = EmployeeModel.empty();
  DateTime _selectedDate = DateTime.now();

  SalaryModel get salary => _selectedSalary;
  DateTime get selectedDate => _selectedDate;
  EmployeeModel get employee => selectedEmployee;

  SalaryViewModel({required SalaryService salaryService}) : _salaryService = salaryService;

  void setEmployee(EmployeeModel employee) {
    selectedEmployee = employee;
    notifyListeners();
  }

  void setDate(DateTime date) {
    _selectedDate = date;
    notifyListeners();
  }

  Future<void> getSalary() async {
```

```
    try {
      int employeeID = selectedEmployee.id;
      DateTime date = selectedDate;
      String period = date.year.toString() + date.month.toString().padLeft(2, '0');
      Map<String, dynamic> data1 = await _salaryService.getEmployeeSalary(employeeID,
period);

      if (data1.isEmpty) {
        String startDate = DateFormat('yyyy-MM-dd').format(DateTime(date.year, date.month -
1, 1));
        String endDate = DateFormat('yyyy-MM-dd').format(DateTime(date.year, date.month,
1));
        List<dynamic> data2;
        if (selectedEmployee.position.type == 1) {
          data2 = await _salaryService.getWorkerSalary(employeeID, startDate, endDate);
        } else {
          data2 = await _salaryService.getLaborSalary(employeeID, startDate, endDate);
        }
        _selectedSalary = SalaryModel.fromFunction(data2, period, employeeID);
      } else {
        _selectedSalary = SalaryModel.fromMap(data1);
      }

      notifyListeners();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Get Salary error: ${e.message}');
      } else {
        debugPrint('Get Salary error: ${e.toString()}');
      }
    }
  }

  Future<void> storeSalary() async {
    try {
      await _salaryService.store(selectedEmployee, _selectedSalary);

      notifyListeners();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Store Salary error: ${e.message}');
      } else {
        debugPrint('Store Salary error: ${e.toString()}');
      }
    }
  }

  Future<void> updateSalary() async {
    try {
      await _salaryService.update(selectedEmployee, _selectedSalary);

      notifyListeners();
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Update Salary error: ${e.message}');
      } else {
        debugPrint('Update Salary error: ${e.toString()}');
      }
    }
  }
}
```

Segmen Program 4.22 SQL *function* menghitung gaji karyawan

```
create function calculate_salary(emp_id integer, start_date date, end_date date) returns
TABLE(
    total_working integer,
    total_salary numeric,
    total_overtime numeric,
    daily_wage numeric,
    total_attendance integer,
    total_cut numeric,
    monthly_wage numeric
) language plpgsql as $ $ DECLARE total_salary NUMERIC;

total_attendance INT;

total_cut NUMERIC;

monthly_wage NUMERIC;

daily_wage NUMERIC;

total_leave INT;

total_half_attendance INT;

total_working_day INT;

overtime_rate NUMERIC;

total_overtime_hour INT;

total_overtime NUMERIC;

BEGIN -- Initialize variables
total_salary := 0;

monthly_wage := 0;

daily_wage := 0;

total_attendance := 0;

total_leave := 0;

total_half_attendance := 0;

total_working_day := 0;

overtime_rate := 0;

total_overtime_hour := 0;

SELECT
    gaji_pokok INTO monthly_wage
FROM
    karyawan
WHERE
    id_karyawan = emp_id;

SELECT
    value INTO overtime_rate
FROM
    konfigurasi
WHERE
    nama_konfigurasi = 'uang_lembur';
```

```
SELECT
    ROUND(
        COALESCE(
            SUM(
                EXTRACT(
                    EPOCH
                    FROM
                        (waktu_akhir - waktu_mulai)
                ) / 3600
            ),
            0
        ),
        0
    ) into total_overtime_hour
FROM
    izin
WHERE
    id_karyawan = emp_id
    AND jenis = 7
    AND waktu_mulai BETWEEN start_date
    AND end_date;

SELECT
    count(*) INTO total_attendance
FROM
    absensi
WHERE
    id_karyawan = emp_id
    AND waktu_masuk BETWEEN start_date
    AND end_date;

SELECT
    count(*) INTO total_half_attendance
FROM
    izin
WHERE
    id_karyawan = emp_id
    AND (
        jenis = 1
        OR jenis = 2
    )
    AND waktu_mulai BETWEEN start_date
    AND end_date
    AND id_approver IS NOT NULL;

WITH RECURSIVE dates AS (
    SELECT
        start_date AS date
    UNION
    ALL
    SELECT
        (date + INTERVAL '1 day') :: DATE
    FROM
        dates
    WHERE
        date + INTERVAL '1 day' < end_date
)
SELECT
    COUNT(*) INTO total_working_day
FROM
    dates
    LEFT JOIN hari_libur ON dates.date = hari_libur.tanggal
WHERE
    EXTRACT(
        DOW
        FROM
```

```
            dates.date
    ) != 0
    AND hari_libur.tanggal IS NULL;

daily_wage := CASE
    WHEN total_working_day = 0 THEN 0
    ELSE monthly_wage / total_working_day
END;

total_cut := (daily_wage / 2) * total_half_attendance;

total_overtime := total_overtime_hour * overtime_rate;

total_salary := (daily_wage * total_attendance) + total_overtime - total_cut;

RETURN QUERY
SELECT
    total_working_day,
    total_salary,
    total_overtime,
    daily_wage,
    total_attendance,
    total_cut,
    monthly_wage;

END;

$ $;
```

Segmen Program 4.23 SQL *function* menghitung gaji cabutan

```
create function calculate_salary_labor(emp_id integer, start_date date, end_date date)
returns TABLE(
    total_attendance integer,
    total_salary numeric,
    total_overtime numeric,
    total_gram numeric,
    food_allowance numeric
) language plpgsql as $ $ DECLARE total_salary NUMERIC;

total_overtime NUMERIC;

overtime_rate NUMERIC;

total_overtime_hour INT;

total_attendance INT;

total_gram NUMERIC;

gram_wage NUMERIC;

food_allowance NUMERIC;

BEGIN gram_wage := 0;

total_attendance := 0;

total_gram := 0;

food_allowance := 0;

SELECT
    CAST(value AS FLOAT) INTO gram_wage
FROM
    konfigurasi
```

```
WHERE
    nama_konfigurasi = 'gaji_produksi_gram';

SELECT
    value INTO overtime_rate
FROM
    konfigurasi
WHERE
    nama_konfigurasi = 'uang_lembur';

SELECT
    ROUND(
        COALESCE(
            SUM(
                EXTRACT(
                    EPOCH
                    FROM
                        (waktu_akhir - waktu_mulai)
                ) / 3600
            ),
            0
        ),
        0
    ) into total_overtime_hour
FROM
    izin
WHERE
    id_karyawan = emp_id
    AND jenis = 7
    AND waktu_mulai BETWEEN start_date
    AND end_date;

SELECT
    count(*) INTO total_attendance
FROM
    absensi
WHERE
    id_karyawan = emp_id
    AND waktu_masuk BETWEEN start_date
    AND end_date;

SELECT
    sum(berat_awal) INTO total_gram
FROM
    absensi_detail
    JOIN absensi ON absensi_detail.id_absensi = absensi.id_absensi
    JOIN public.karyawan k ON k.id_karyawan = absensi.id_karyawan
WHERE
    k.id_karyawan = emp_id
    AND absensi.waktu_masuk BETWEEN start_date
    AND end_date;

SELECT
    CAST(value AS FLOAT) INTO food_allowance
FROM
    konfigurasi
WHERE
    nama_konfigurasi = 'uang_makan_cabutan';

total_salary := (gram_wage / 10) * total_gram;

food_allowance = food_allowance * total_attendance;

total_overtime := total_overtime_hour * overtime_rate;

RETURN QUERY
SELECT
```

```
    total_attendance,
    total_salary,
    total_overtime,
    total_gram,
    food_allowance;

END;

$ $;
```

4.2.8. Help Chat

Segmen Program 4.24 *Model chat*

```
class ChatModel {
  final int id;
  final int unreadCount;
  final UserModel user;
  final MessageModel lastMessage;

  ChatModel({
    required this.id,
    required this.unreadCount,
    required this.user,
    required this.lastMessage,
  });

  factory ChatModel.fromMap(Map<String, dynamic> chat) {
    return ChatModel(
      id: chat['id_chat'],
      unreadCount: chat['jumlah_belum_dibaca'],
      user: UserModel.fromSupabase(chat['user'][0]),
      lastMessage: MessageModel.fromSupabase(chat['pesan'][0]),
    );
  }

  static List<ChatModel> fromMapList(List<Map<String, dynamic>> chats) {
    return chats.map((chat) => ChatModel.fromMap(chat)).toList();
  }
}
```

Segmen Program 4.25 *Model* pesan

```
class MessageModel {
  final int id;
  final String userId;
  final String? message;
  final String? filePath;
  final File? file;
  final bool isAdmin;
  final DateTime timestamp;

  MessageModel({
    required this.id,
    required this.userId,
    this.message,
    this.file,
    this.filePath,
    required this.isAdmin,
    required this.timestamp,
  });

  factory MessageModel.fromSupabase(Map<String, dynamic> message) {
```

```
    return MessageModel(
      id: message['id_pesan'],
      userId: message['id_user'],
      message: message['pesan'],
      isAdmin: message['is_admin'],
      timestamp: DateTime.parse(message['waktu_kirim']),
    );
  }

  static List<MessageModel> fromSupabaseList(List<Map<String, dynamic>> messages) {
    return messages.map((message) => MessageModel.fromSupabase(message)).toList();
  }
}
```

Segmen Program 4.26 *Service chat*

```
class ChatService {
  SupabaseClient supabase = Supabase.instance.client;

  SupabaseStreamBuilder getAdminMessage() {
    try {
      final stream = supabase.from('chat').stream(primaryKey: ['id_chat']);

      return stream;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  SupabaseStreamBuilder getUserMessage(int chatID) {
    try {
      final stream = supabase.from('pesan').stream(primaryKey: ['id_pesan']).eq('id_chat',
chatID).order('waktu_kirim', ascending: false);

      return stream;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<List<Map<String, dynamic>>> getAdminChatList(List<dynamic> chatIds) async {
    try {
      final data = await supabase.from('chat').select('''
        *,
        user!inner(*,karyawan!inner(*, posisi!inner(*))),
        pesan!inner(*)
      ''').inFilter('id_chat', chatIds).limit(1, referencedTable:
'pesan').order('waktu_kirim', referencedTable: 'pesan', ascending: false);

      return data;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<void> sendMessage(int chatID, String userID, bool isAdmin, {String? message, File?
file}) async {
    try {
```

```
      String? path;
      if (file != null) {
        String fileName = file.path.split('/').last;
        path = await supabase.storage.from('chat').upload(
              '$chatID/$fileName',
              file,
              fileOptions: const FileOptions(cacheControl: '3600', upsert: false),
            );
      }

      await supabase.from('pesan').insert({
        'id_chat': chatID,
        'id_user': userID,
        'is_admin': isAdmin,
        'pesan': message,
        'file': path,
        'waktu_kirim': DateTime.now().toIso8601String(),
      });
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }

  Future<Uint8List> downloadFile(String path) async {
    try {
      String bucketID = path.split('/')[0];
      String objectPath = path.split('/').sublist(1).join('/');
      Uint8List file = await supabase.storage.from(bucketID).download(objectPath);
      return file;
    } on PostgrestException catch (error) {
      throw PostgrestException(message: error.message);
    } catch (e) {
      throw Exception(e.toString());
    }
  }
}
```

Segmen Program 4.27 *ViewModel chat*

```
class ChatViewModel extends ChangeNotifier {
  final ChatService _chatService;
  ChatModel _selectedChat = ChatModel.empty();
  MessageModel _selectedMessage = MessageModel.empty();

  ChatViewModel({required ChatService chatService}) : _chatService = chatService;

  ChatModel get selectedChat => _selectedChat;
  MessageModel get selectedMessage => _selectedMessage;
  set selectedChat(ChatModel chat) {
    _selectedChat = chat;
    notifyListeners();
  }

  set selectedMessage(MessageModel message) {
    _selectedMessage = message;
    notifyListeners();
  }

  SupabaseStreamBuilder? getMessageStream(int chatID) {
    try {
      final stream = _chatService.getUserMessage(chatID);
```

```
      return stream;
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Get Message error: ${e.message}');
      } else {
        debugPrint('Get Message error: ${e.toString()}');
      }
      return null;
    }
  }

  SupabaseStreamBuilder? getChatStream() {
    try {
      final stream = _chatService.getAdminMessage();

      return stream;
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Get Message error: ${e.message}');
      } else {
        debugPrint('Get Message error: ${e.toString()}');
      }
      return null;
    }
  }

  Future<List<ChatModel>> getChat(List<Map<String, dynamic>> chats) async {
    try {
      List<Map<String, dynamic>> data = await _chatService.getAdminChatList(chats.map((e)
=> e['id_chat']).toList());

      return ChatModel.fromMapList(data);
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Get Chat: ${e.message}');
      } else {
        debugPrint('Get Chat error: ${e.toString()}');
      }
      return [];
    }
  }

  Future<void> sendMessage(int chatID, String userID, bool isAdmin, {String? meesage, File?
file}) async {
    try {
      _chatService.sendMessage(chatID, userID, isAdmin, message: meesage, file: file);
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Send Message error: ${e.message}');
      } else {
        debugPrint('Send Message error: ${e.toString()}');
      }
    }
  }

  Future<void> downloadFile() async {
    try {
      String path = _selectedMessage.filePath!;
      String fileName = path.split('/').last;
      Uint8List file = await _chatService.downloadFile(path);

      File(fileName).writeAsBytes(file, flush: true);
    } catch (e) {
      if (e is PostgrestException) {
        debugPrint('Download File error: ${e.message}');
      } else {
        debugPrint('Download File error: ${e.toString()}');
```

```
        }
      }
    }
}
```